**Directory Organization for the Homework Assignment**

1. **Base Directory Structure**

c

Copy code

hw2\_id1\_id2/

├── src/

│ ├── main.c

│ ├── dispatcher.c

│ ├── dispatcher.h

│ ├── worker.c

│ ├── worker.h

│ ├── queue.c

│ ├── queue.h

│ ├── utils.c

│ ├── utils.h

├── include/

│ ├── common.h

├── tests/

│ ├── test.txt

│ └── other\_tests.txt

├── output/

│ ├── counter files/

│ ├── log files/

│ ├── stats.txt

├── Makefile

├── hw2\_id1\_id2.pdf

└── README.md

1. **Files and Their Contents**
   * **src/main.c**
     + Contains the main function.
     + Parses command-line arguments.
     + Initializes the dispatcher and worker threads.
     + Manages cleanup and final statistics.
     + Keeps track of program start time.
   * **src/dispatcher.c**
     + Contains the dispatcher logic.
     + Reads and parses the input command file.
     + Handles dispatcher commands (msleep, wait).
     + Queues jobs for workers.
     + Writes to the dispatcher log file.
   * **src/dispatcher.h**
     + Declares functions for the dispatcher logic.
   * **src/worker.c**
     + Implements worker thread functionality.
     + Executes jobs from the shared queue.
     + Handles basic commands (msleep, increment, decrement, repeat).
     + Manages worker-specific log files.
   * **src/worker.h**
     + Declares functions for worker logic.
   * **src/queue.c**
     + Implements a thread-safe queue for the shared work queue.
     + Handles synchronization between dispatcher and worker threads.
   * **src/queue.h**
     + Declares queue-related data structures and functions.
   * **src/utils.c**
     + Contains utility functions (e.g., time calculation, file operations).
     + Manages counter file updates (increment, decrement).
     + Provides common synchronization functions.
   * **src/utils.h**
     + Declares utility functions and shared constants.
   * **include/common.h**
     + Contains global constants and shared macros (e.g., max line width, max threads, max counters).
     + Defines shared data structures, such as the job structure.
   * **tests/test.txt**
     + A sample command file for testing.
     + Includes commands to test all dispatcher and worker functionality.
   * **output/**
     + Subdirectories for counter files (countxx.txt), log files (threadxx.txt, dispatcher.txt), and stats.txt.
   * **Makefile**
     + Builds the executable (hw2).
     + Includes targets: all, clean, test.
   * **hw2\_id1\_id2.pdf**
     + External documentation describing the implementation, challenges, and design decisions.
   * **README.md**
     + Provides an overview of the project, instructions for building and running the program, and examples.

**Work Plan for Implementation**

1. **Setup and Initialization**
   * **Goal:** Create the project structure and basic stubs for all files.
   * **Steps:**
     1. Set up the directory structure.
     2. Write the Makefile to compile all source files.
     3. Write initial versions of main.c, dispatcher.c, and worker.c with placeholder functions.
2. **Core Dispatcher Logic**
   * **Goal:** Implement the dispatcher logic for sequential commands and job queuing.
   * **Steps:**
     1. Parse command-line arguments in main.c.
     2. Implement the dispatcher command handling (msleep, wait) in dispatcher.c.
     3. Add logic to read the command file and queue jobs.
3. **Thread-Safe Work Queue**
   * **Goal:** Implement a thread-safe queue to manage jobs.
   * **Steps:**
     1. Define the queue data structure in queue.h.
     2. Implement enqueue, dequeue, and synchronization in queue.c.
     3. Test the queue with a simple producer-consumer model.
4. **Worker Thread Implementation**
   * **Goal:** Implement worker threads to execute queued jobs.
   * **Steps:**
     1. Create worker threads in main.c during initialization.
     2. Implement job handling in worker.c, including parsing and executing basic commands.
     3. Use pthread\_mutex and pthread\_cond for synchronization.
5. **File Operations**
   * **Goal:** Manage counter files and ensure atomic updates.
   * **Steps:**
     1. Create counter files during initialization.
     2. Implement atomic increment and decrement in utils.c.
     3. Ensure thread safety using file locks or synchronization primitives.
6. **Logging and Timing**
   * **Goal:** Add logging for workers and the dispatcher.
   * **Steps:**
     1. Record start and end times for jobs.
     2. Write log messages to the appropriate files.
     3. Use gettimeofday or clock\_gettime to calculate timestamps.
7. **Statistics Collection**
   * **Goal:** Collect and output job statistics.
   * **Steps:**
     1. Track job turnaround times.
     2. Compute total, min, max, and average turnaround times.
     3. Write the statistics to stats.txt.
8. **Testing and Debugging**
   * **Goal:** Ensure correctness and handle edge cases.
   * **Steps:**
     1. Write comprehensive test cases in tests/test.txt.
     2. Validate against edge cases (e.g., invalid commands, zero counters, max threads).
     3. Debug and optimize synchronization to avoid deadlocks.
9. **Documentation and Submission**
   * **Goal:** Complete external documentation and prepare the submission package.
   * **Steps:**
     1. Write the PDF documentation (hw2\_id1\_id2.pdf).
     2. Test the program and verify submission requirements.
     3. Package the solution into a .tgz file for submission.

This structured approach ensures modularity, correctness, and maintainability.